```
# Chris Straschewski
# CMS180014
# CS 4395.001


# 1. Create a Python notebook with appropriate headings, which you will print-to-pdf to upload.
# Write a 2-3 sentence summary of WordNet.

# WordNet:
# WordNet is a lexical database containing nouns, verbs, adjectives and adverbs in a hierarchical
# fashion. Some of the features of WordNet include glosses, (short definitions) synsets, (synonym sets)
# and relations between words. Each of these features can have several methods applied upon them.
# For example, regarding synsets, there are the definition(), (retrieves the gloss) examples(),
# (gives usage cases) and lemmas() (returns a list of WordNet entries that are synonyms) methods.
# Overall, WordNet is quite useful for finding relations and groupings between words and has plenty
# of practical uses.


# 2. Select a noun. Output all synsets.

# import WordNet
import nltk
from nltk.corpus import wordnet as wn
# select a noun and output all synsets of it
# I chose "book" and made a for loop to iterate through all the synsets
print("All synsets of 'book':")
for synset in wn.synsets('book'):
    print(synset.name())
print()
# Output:
# book.n.01
# book.n.02
# record.n.05
# script.n.01
# ledger.n.01
# book.n.06
# book.n.07
# koran.n.01
# bible.n.01
# book.n.10
# book.n.11
# book.v.01
# reserve.v.04
# book.v.03
# book.v.04


# 3. Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas.
# From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the
# synsets as you go. Write a couple of sentences observing the way that WordNet is organized for
# nouns.

# I choose "record.n.05" as my synset
# extract definition
definition = wn.synset('record.n.05').definition()
print("Definition:", definition, "\n")
# Output: a compilation of the known facts regarding something or someone
```

```
# usage examples
examples = wn.synset('record.n.05').examples()
print("Examples:", examples, "\n")
# Output: ["Al Smith used to say, `Let's look at the record'", 'his name is in all the record books']

# lemmas
lemmas = wn.synset('record.n.05').lemmas()
print("Lemmas:", lemmas, "\n")
# Output: [Lemma('record.n.05.record'), Lemma('record.n.05.record_book'), Lemma('record.n.05.book')]

# traverse up from 'record' synset
record = wn.synset('record.n.05')
hyper = lambda s: s.hypernyms()
hierarchy_list = list(record.closure(hyper))
print("Hierarchy:", hierarchy_list, "\n")
# Output: [Synset('fact.n.02'), Synset('information.n.01'), Synset('message.n.02'),
# Synset('communication.n.02'), Synset('abstraction.n.06'), Synset('entity.n.01')]

# it's clear that the nouns are ordered in a hierarchical fashion, as you read, the terms get broader
# and broader, from "facts" and "information", all the way to "entity"


# 4. Output the following (or an empty list if none exist): hypernyms, hyponyms, meronyms,
# holonyms, antonym.

# hypernyms
hypernyms = wn.synset('record.n.05').hypernyms()
print("Hypernyms:", hypernyms, "\n")
# Output: [Synset('fact.n.02')]

# hyponyms
hyponyms = wn.synset('record.n.05').hyponyms()
print("Hyponyms:", hyponyms, "\n")
# Output: [Synset('card.n.08'), Synset('logbook.n.01'), Synset('won-lost_record.n.01')]

# meronyms
meronyms = wn.synset('record.n.05').part_meronyms()
print("Meronyms:", meronyms, "\n")
# Output: null

# holonyms
holonyms = wn.synset('record.n.05').part_holonyms()
print("Holonyms:", holonyms, "\n")
# Output: null

# antonyms
antonym = wn.synset('record.n.05').lemmas()[0].antonyms()
print("Antonyms:", antonym, "\n")
# Output: null


# 5. Select a verb. Output all synsets.
# I choose "have" as my verb
print("All synsets of 'have':")
for synset in wn.synsets('have'):
    print(synset.name())
print()
# Output:
```

```
# Output:
# rich_person.n.01
# have.v.01
# have.v.02
# experience.v.03
# own.v.01
# get.v.03
# consume.v.02
# have.v.07
# hold.v.03
# have.v.09
# have.v.10
# have.v.11
# have.v.12
# induce.v.02
# accept.v.02
# receive.v.01
# suffer.v.02
# have.v.17
# give_birth.v.01
# take.v.35


# 6. Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas.
# From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the
# synsets as you go. Write a couple of sentences observing the way that WordNet is organized for
# verbs.

# I choose "consume.v.2" as my synset
# extract definition
definition_v = wn.synset('consume.v.2').definition()
print("Definition:", definition_v, "\n")
# Output: serve oneself to, or consume regularly

# usage examples
examples_v = wn.synset('consume.v.2').examples()
print("Examples:", examples_v, "\n")
# Output: ['Have another bowl of chicken soup!', "I don't take sugar in my coffee"]

# lemmas
lemmas_v = wn.synset('consume.v.2').lemmas()
print("Lemmas:", lemmas_v, "\n")
# Output: [Lemma('consume.v.02.consume'), Lemma('consume.v.02.ingest'),
# Lemma('consume.v.02.take_in'), Lemma('consume.v.02.take'), Lemma('consume.v.02.have')]

# traverse up from 'consume' synset
consume_v = wn.synset('consume.v.2')
hyper_v = lambda x: x.hypernyms()
hierarchy_list_v = list(consume_v.closure(hyper))
print("Hierarchy:", hierarchy_list_v, "\n")
# Output: null


# 7. Use morphy to find as many different forms of the word as you can.
print(wn.morphy('consume', wn.ADJ))

print(wn.morphy('consume', wn.VERB))

print(wn.morphy('consume', wn.NOUN))
```

```
print(wn.morphy('consume'))
# Outputs for these:
# None
# consume
# None
# consume


# 8. Select two words that you think might be similar. Find the specific synsets you are interested in.
# Run the Wu-Palmer similarity metric and the Lesk algorithm. Write a couple of sentences with
# your observations.

# "tall" and "long"

# Specific synsets
print("\nAll synsets of 'tall':")
for synset in wn.synsets('tall'):
    print(synset.name(), ":", wn.synset(synset.name()).definition())
print()

print("All synsets of 'long':")
for synset in wn.synsets('long'):
    print(synset.name(), ":", wn.synset(synset.name()).definition())

# All synsets of 'tall':
# tall.n.01 : a garment size for a tall person
# tall.a.01 : great in vertical dimension; high in stature
# grandiloquent.s.01 : lofty in style
# tall.s.03 : impressively difficult
# improbable.s.03 : too improbable to admit of belief

# All synsets of 'long':
# hanker.v.01 : desire strongly or persistently
# long.a.01 : primarily temporal sense; being or indicating a relatively great or greater than average duration or passage of time or a duration as specified
# long.a.02 : primarily spatial sense; of relatively great or greater than average spatial extension or extension as specified
# long.s.03 : of relatively great height; - Sherwood Anderson
# retentive.a.01 : good at remembering
# long.a.05 : holding securities or commodities in expectation of a rise in prices
# long.a.06 : (of speech sounds or syllables) of relatively long duration
# long.s.07 : involving substantial risk
# farseeing.s.02 : planning prudently for the future
# long.s.09 : having or being more than normal or necessary:
# long.r.01 : for an extended time or at a distant time
# long.r.02 : for an extended distance

# I choose these two:
# tall.a.01 : great in vertical dimension; high in stature
# long.a.02 : primarily spatial sense; of relatively great or greater than average spatial extension or extension as specified

# Wu-Palmer
tall = wn.synset('tall.a.01')
long = wn.synset('long.a.02')
print("\nWu-Palmer Similarity:", wn.wup_similarity(tall, long))
# Wu-Palmer Similarity: 0.5

# Lesk Algorithm
from nltk.wsd import lesk
```

```
sent = ['The', 'man', 'was', 'very', 'big', 'and', 'tall', '.']
print("\n", lesk(sent, 'tall', 'a'))
# Synset('tall.a.01')

sent2 = ['The', 'man', 'had', 'very', 'long', 'legs', '.']
print("\n", lesk(sent2, 'long', 'a'))
print()
# Synset('long.a.06')

# I expected the Wu-Palmer similarity metric to get a higher score than that. I think tall and long
# are quite similar. However, .5 isn't that bad.
# The Lesk Algorithm worked as expected for tall, but found the wrong word for long.
# If I tried more sentences, I could likely get it to work correctly though.


# 9. Write a couple of sentences about SentiWordNet, describing its functionality and possible use
# cases. Select an emotionally charged word. Find its senti-synsets and output the polarity scores
# for each word. Make up a sentence. Output the polarity for each word in the sentence. Write a
# couple of sentences about your observations of the scores and the utility of knowing these

# SentiWordNet is a feature built on top of WordNet that attempts to evaluate a sentence based on its
# levels of positivity, negativity, and objectivity. For example, if I wrote "I hate you."
# it should come out more negative than if I wrote "I love you."

# Emotionally charged word: "hate"
# Find senti-synsets and output polarity scores for each word

from nltk.corpus import sentiwordnet as swn
# senti_synsets
senti_list = list(swn.senti_synsets('hate'))
for item in senti_list:
    print(item)
# <hate.n.01: PosScore=0.125 NegScore=0.375>
# <hate.v.01: PosScore=0.0 NegScore=0.75>

# polarity scores
hate1 = swn.senti_synset('hate.n.01')
print("\nSenti_synsets:", hate1)
print("Positive score = ", hate1.pos_score())
print("Negative score = ", hate1.neg_score())
print("Objective score = ", hate1.obj_score())
# Senti_synsets: <hate.n.01: PosScore=0.125 NegScore=0.375>
# Positive score =  0.125
# Negative score =  0.375
# Objective score =  0.5

hate2 = swn.senti_synset('hate.v.01')
print("\nSenti_synsets:", hate2)
print("Positive score = ", hate2.pos_score())
print("Negative score = ", hate2.neg_score())
print("Objective score = ", hate2.obj_score())
print("\n")
# Senti_synsets: <hate.v.01: PosScore=0.0 NegScore=0.75>
# Positive score =  0.0
# Negative score =  0.75
# Objective score =  0.25

# Make up a sentence
# "I hate friends!"
```

```
# I hate friends.
# Output the polarity for each word in the sentence.
senti_list = list(swn.senti_synsets('I'))
for item in senti_list:
    print(item)
print()

senti_list = list(swn.senti_synsets('hate'))
for item in senti_list:
    print(item)
print()

senti_list = list(swn.senti_synsets('friends'))
for item in senti_list:
    print(item)
print()

# Scores
I = swn.senti_synset('i.n.03')
print("\nSenti_synsets:", I)
print("Positive score = ", I.pos_score())
print("Negative score = ", I.neg_score())
print("Objective score = ", I.obj_score())

hate = swn.senti_synset('hate.v.01')
print("\nSenti_synsets:", hate)
print("Positive score = ", hate.pos_score())
print("Negative score = ", hate.neg_score())
print("Objective score = ", hate.obj_score())

friends = swn.senti_synset('friend.n.01')
print("\nSenti_synsets:", friends)
print("Positive score = ", friends.pos_score())
print("Negative score = ", friends.neg_score())
print("Objective score = ", friends.obj_score())

# The scores are not surprising in the slightest, except that friends isn't a bit more positive.
# These scores would be useful in an NLP application because you could determine a lot by
# knowing if a sentence is positive or negative.


# 10. Write a couple of sentences about what a collocation is. Output collocations for text4, the
# Inaugural corpus. Select one of the collocations identified by NLTK. Calculate mutual
# information. Write commentary on the results of the mutual information formula and your
# interpretation.

# Collocations are a grouping of words that often appear together, and if you try to substitute
# any of the words, the meaning changes. Example: "try hard" and "try difficult" are not the same

# import text4
from nltk.book import *
text4

# get collocations
text4.collocations()
# United States; fellow citizens; years ago; four years; Federal
# Government; General Government; American people; Vice President; God
# bless; Chief Justice; one another; fellow Americans; Old World;
# Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
```

```
# tribes; public debt; foreign nations

# I will use United States

# Calculate Mutual Information
# Mutual Information = P(x,y) / [P(x) * P(y)]

text = ' '.join(text4.tokens)

import math
vocab = len(set(text4))
ab = text.count('United States')/vocab
print("\np(United States) = ",ab )
a = text.count('United')/vocab
print("p(United) = ", a)
b = text.count('States')/vocab
print('p(States) = ', b)
pmi = math.log2(ab / (a * b))
print('PMI = ', pmi)
# p(United States) =  0.015860349127182045
# p(United) =  0.0170573566084788
# p(States) =  0.03301745635910224
# PMI =  4.815657649820885

# The Mutual Information score is fairly high, indicating "United States" is likely a collocation,
# which makes sense because it is a title/name. Changing the words would change the meaning because
# you wouldn't be talking about the USA anymore. PMI seems to be an accurate algorithm.
```